Notes on computing peaks in k-levels and parametric spanning trees

AUTHOR(S):

Katoh, Naoki; Tokuyama, Takeshi

CITATION:

Katoh, Naoki ...[et al]. Notes on computing peaks in k-levels and parametric spanning trees. Proceedings of the seventeenth annual symposium on Computational geometry 2001: 241-248

ISSUE DATE:

2001

URL:

http://hdl.handle.net/2433/84853

Notes on computing peaks in a k-level and a parametric spanning tree

Naoki Katoh* Takeshi Tokuyama†

Abstract

We give an algorithm to compute all the local peaks in the k level of an arrangement of n lines in $\tilde{O}((kn)^{2/3}+n)$ time. We can also find τ largest peaks in $\tilde{O}((\tau n)^{2/3}+n)$ time. Moreover, we consider the longest edge in a parametric minimum spanning tree, and give an algorithm to compute the parameter value (within a given interval) maximizing/minimizing the length of the longest edge in MST. The time complexity is $\tilde{O}(n^{8/7}k^{1/7}+nk^{1/3})$.

1 Introduction

The k-level of an arrangement of lines is one of popular geometric objects in computational geometry [7, 21]. The k-level is the union of k-th lowest (closed) edges of the arrangement, and it can be considered as the trajectory of the k-th smallest element in a set of n data that depends on a parameter t linearly. Thus, the k-level is a special case of the locus of the largest element of the minimum base of a parametric matroid [11, 12, 8, 14]; in precise, the k-level is the locus of the maximum element in the minimum base of a parametric uniform matroid of rank k. It is known that the complexity $g_k(n)$ of the k-level of an arrangement of n lines is $O(k^{1/3}n)$ [6] and $2^{\Omega(\sqrt{k})}n$ [20]. The same upper bound holds for any parametric matroid (with a linear parameter) of rank k in n elements. Moreover, it is known that the k-level can be computed in $g_k(n)\log^2 n$.

However, we often need a compact "outline" of the k-level as a trajectory of a parametric problem by using characteristic points on it. Peaks in the trajectory are considered to be characteristic points naturally. The key observation to investigate the k-level is that it has at most $2k-1$ local peaks (at most k maximal peaks and at most $k-1$ minimal peaks) [1, 8]. One interesting question is how we can compute all the local peaks efficiently. This enables us to give a decomposition of k-level into monotone chains, and hence create an outline of the k-level. We give an algorithm to compute all the local peaks in the k level of an arrangement of n lines in $\tilde{O}((kn)^{2/3}+n)$ time, where $\tilde{O}$ is the big-O notation ignoring polylogarithmic factors. Although it is possible to give exact polylogarithmic factors, we omit them in order to improve the readability. Such an outline, generally speaking, will be useful to design a compact information to control parametric problems, and possibly utilized in designing kinetic data structures [2].

*Department of Architecture, Kyoto University
†Graduate School of Information Sciences, Tohoku University, tokuyama@dais.is.tohoku.ac.jp

Another interesting question is how fast we can compute τ largest maximal peaks for $\tau \leq k$. If $\tau = 1$, Roos and Widmayer [18] gave a neat method to compute the maximum point in the k-level in $O(n \log n + (n-k)\log^2(n-k))$ time by using an efficient slope selection algorithm. We can compute τ largest peaks in $\tilde{O}((\tau n)^{2/3} + n)$ time by combining Roos and Widmayer's technique and the above mentioned method for computing all the peaks.

Finally, we investigate whether we can analogously treat some parametric matroids: Compute peaks in the trajectory of the largest element in the minimum weight base of a parametric matroid. In particular, the graphic matroid is of wide interest: Given a weighted undirected graph $G(x)$ with k nodes and n edges, such that each edge has an edge function which is linear in a parameter x. Let $T(x)$ be the minimum weight spanning tree of $G(x)$ and consider the longest edge $e(x)$ in $T(x)$. Note that the minimum weight spanning tree becomes a tree that minimizes the length of the longest edge. We call the edge $e(x)$ the *spanning bottleneck edge*(SBE), and denote $SBE(x)$ and $w_{SBE}(x)$ for $e(x)$ and its weight, respectively. The naming comes from the fact that we do not have a connected spanning graph of $G(x)$ if we only have edges with less weights than $w_{SBE}(x)$. Here, k and n becomes the rank and size of the graphic matroid, respectively.

The following problems are important in sensitivity analysis. (1): Compute the maximum value and the minimum value of $w_{SBE}(x)$ for $x \in I$, where I is a given interval. (2): Compute all peaks of the trajectory $y = w_{SBE}(x)$.

Both problems can be solved by computing the whole picture of the transitions of the minimum spanning trees, and the time complexity of the current best algorithm is $O(kn \log n)$ [10]. Roos and Widmayer's method can be directly applied to the first problem. By using dynamic maintenance algorithms [9] of a minimum spanning tree, the time complexity becomes $O(\sqrt{kn}\log^2 n)$. Combined with range searching techniques, we improve the time complexity to $\tilde{O}(n^{8/7}k^{1/7} + nk^{1/3})$. We give some discussion on the second problem, although theoretical improvement on the $O(kn \log n)$ time method is remained open.

2 Preliminaries

2.1 Roos and Widamyer's algorithm

Given a set $\mathcal{H}$ of n lines in the x-y plane, let $\mathcal{L}_k$ be the k-level of the arrangement of $\mathcal{H}$. Let p be a point on $\mathcal{L}_k$ that have the maximum y-value y_{max}. Without loss of generality, we assume that such a point is unique.

For any given value α, one can decide whether $y_{max} \geq \alpha$ or not in $O(n \log n)$ time: We sweep on the line $h : y = \alpha$ from left to the right to compute the levels of all intersection points on h with lines in the arrangement. By using this decision method, a binary search algorithm works to compute p, and a weakly polynomial time algorithm with a time complexity $O(n \log n \log \Gamma)$ can be obtained, if each coefficient of the lines is an integer with $\log \Gamma$ bits. Roos and Widamyer[18] applied the slope selection method to transform the binary search algorithm into strongly polynomial, and gave an $O(n \log^2 n)$ time algorithm. They further improved the time complexity to $O(n \log n + k \log^2 k)$ for computing the minimum and $O(n \log n + (n-k)\log^2(n-k))$ for computing the maximum.

2.2 Range query and Matoušek's point set decomposition

We use well-known (but sophisticated) half-space range query data structures: We construct a data structure for a set S of n points in a plane such that given a query halfplane H, we can compute the number of points in S located in H efficiently. If we spend $O(m)$ time for constructing the data space for $n \log n \leq m \leq n^2$, the query time is $\tilde{O}(n/m^{1/2})$. The query can be done in polylogarithmic time by using $O(n/m^{1/2})$ processors. Moreover, we can query the number of points in the intersection of two (or three) halfspaces in the same query time if we ignore a polylogarithmic factor. We can also do reporting query if we spend additional $O(N)$ time if the region contains N points.

The main building block for the range query is the point set decomposition structure of Matoušek, which we also need to utilize directly (we only describe its two-dimensional version):

Theorem 2.1 (Matoušek) *Given a set S of n points in the plane, for any given $r < n$, we can subdivide S into r disjoint subsets S_i $(i = 1, 2, \ldots, r)$ such that $|S_i| \leq 2n/r$ satisfying the following condition: Each S_i is enclosed in a triangle σ_i, and any hyperplane in the space cut at most $cr^{1/2}$ triangles among $\sigma_1, \sigma_2, \ldots, \sigma_r$ where c is a constant independent of n and r. Such a decomposition can be constructed in $O(n \log n)$ time.*

Given a set $\mathcal{H}$ of n lines in a plane, we consider the set $\mathcal{D}(\mathcal{H})$ of their dual points: The dual point of a line $y = ax - b$ is (a, b). We construct the Matousek's data structure for $\mathcal{D}(\mathcal{H})$. Given a point $p = (x_0, y_0)$, the set of dual points of lines below p is the set of points in $\mathcal{D}(\mathcal{H})$ located below the line $Y = x_0 X - y_0$, where X and Y correspond to coordinates of the dual plane. Thus, we can compute the level of the point p in the arrangement of n lines by using half-plane range searching. Moreover, we have the highest line below p in the same query time. Also, we can query the number of lines which lies below both of a pair of query points.

3 Computing all peaks in k-level

We assume $k \leq n/2$ for simplicity from now on; if $k > n/2$, replace k by $n - k$ and exchange maximal and minimal in the statements. A key observation for the k-level is that it is a subset of a union of k concave chains: thus, a k-level has at most k maximal peaks and $k - 1$ minimal peaks. We want to compute all the local peaks in a given interval I of the x-coordinate value. We mainly focus on the case where the number of peaks in I is $O(n^{1-\epsilon})$ for some $\epsilon > 0$; naturally, this condition holds if $k = O(n^{1-\epsilon})$. Indeed, our method does not improve previous known time-complexity if $\mathcal{L}_k$ has $\Omega(n)$ peaks in I.

We apply a version of parametric search paradigm [17, 19]. However, before applying the parametric search, we start with a simpler "k-branching binary search" method. Without loss of generality, we assume that no line in the arrangement is horizontal nor vertical.

We prepare two key-subroutines: *one-shot query* and *peak counting*: Let $\boldsymbol{p}(x_0)$ be the point on the k-level at the x-coordinate value x_0. Let $\ell_k^-(x_0)$ (resp. $\ell_k^+(x_0)$ be the line in the k-level at the x-coordinate value $x_0 - \epsilon$ (resp. $x_0 + \epsilon$) for infinitesimally small $\epsilon > 0$. If x_0 is not an x-coordinate value of a vertex on the k-level, $\ell_k^-(x_0) = \ell_k^+(x_0)$. The above operation to compute the point (together with lines containing the point) on $\mathcal{L}_k$ at a given

x-coordinate value is called *one-shot query*. The complexity $q(n,m)$ given in the following lemma is called *one-shot query-time* of the k-level:

Lemma 3.1 *If we preprocess the lines in $\mathcal{H}$ with $O(m)$ time for $n\log n < m < n^2$, given a x-coordinate values x_0, we can compute $\boldsymbol{p}(x_0)$, $\ell_k^-(x_0)$ and $\ell_k^+(x_0)$ in $q(n,m) = \tilde{O}(n/m^{1/2})$ time, and also in polylogarithmic time by using $O(n/m^{1/2})$ processors.*

Proof By using the method given in the preliminary section, we can compute the level of any given point (x_0, y_0) in polylogarithmic time by using $O(n/m^{1/2})$ processors. We now apply parametric searching to have the sequential time bound to compute the point $p(x_0)$. A parametric searching algorithm is usually stated as a sequential algorithm; however, it is naturally a parallel algorithm if we use a parallel decision algorithm and also a parallel sorting algorithm. We remark that if we want to save some polylogarithmic factors, we can avoid using parametric searching; however, we omit in this version. □

The peak-counting is a routine to compute the number of peaks of the k-level in a given interval $J = [x_0, x_1]$ efficiently. The following elementary lemma is essential:

Lemma 3.2 *Let $f(x_0)$ and $f(x_1)$ are numbers of positive-slope lines below or on the k-level at x_0 and x_1, respectively. Then, the number of maximal peaks of $\mathcal{L}_k$ in the interval J is $f(x_0) - f(x_1)$.*

Proof At-most-k-level (the part of the arrangement below $k+1$-level) is a union of k concave chains such that all concave peaks in the chains appear in the k-level [1]. If a concave chain among them has a peak in J, the slope of the chain must be changed from positive to negative. Thus, the number of maximal peaks within J is the difference between the numbers of positive slope lines at two endpoints. □

We remark that $f(x_0) - f(x_1)$ equals the number of positive slope lines intersecting the segment between $\boldsymbol{p}(x_0)$ and $\boldsymbol{p}(x_1)$.

Lemma 3.3 *For a given interval J of x-coordinate value, the number $\kappa(J)$ of peaks of $\mathcal{L}_k$ in J can be computed in $O(q(n,m))$ time (recall that $q(n,m)$ is the one-shot query time) if we preprocess the lines with $O(m)$ time. Also, the number of maximal peaks can be computed in $O(q(n,m))$ time.*

Proof If we construct the dual of range search data structure for the set of lines with positive slopes, the number of positive slope lines below a given point (x_0, y_0) can be computed in $O(q(n,m))$ time. Hence, $f(x_0)$ and $f(x_1)$ can be computed in $O(q(n,m))$ time by using the Lemma 3. The number of minimal peaks is easily computed from the number of maximal peaks and slopes of the k-level at endpoints. □

Now, we can apply a binary search paradigm to design a weakly-polynomial time algorithm. First, for the input interval I, we compute the number of peaks $\kappa = \kappa(I)$ within the interval (Lemma 3). The time complexity for this initialization is negligible, and obviously $\kappa \leq 2k-1$. Next, we construct a data structure for the one-shot query in $O(m)$ time, where the choice of m will be explained later. Suppose that each of coefficients of the equations of lines are quotient numbers of integers in $U = [-\Gamma, \Gamma]$. Then, we apply κ-branching binary search on U to find all peaks; At each stage of the binary search, we have at most κ subintervals which has at least one local peak of $\mathcal{L}_k$ (such a subinterval is called an *active interval*), and we recursively search in active subintervals. Thus, after examining $\kappa\log\Gamma$ candidates of x-coordinate values, we can find all of the peaks.

Proposition 3.4 *All the local peaks of $\mathcal{I}_k$ in the interval I can be computed in $O(\kappa q(n,m)\log\Gamma)$ time.*

To make the complexity into strongly polynomial, we apply parametric search by using the parallel algorithm for the one-shot query given in Lemma 3 as its *guide algorithm*. We run the guide algorithm without inputting the parameter value (in our case, an x-coordinate value), and decide the x-coordinate values of the peaks by using sequential one-shot query and the counting algorithm of Lemma 3 as *decision algorithms*. Usually, parametric searching method solves optimization problems on monotone or convex functions. Here, k-level is neither monotone nor convex, but it consists of κ monotone fragments. Thus, while running the guide algorithm, there are at most κ different critical parameter values to determine all the comparisons in the current parallel step that are necessary to proceed into the next parallel step. In precise, the number of different choices is the number of active intervals obtained breaking I by the critical parameter values found so far in the guide algorithm. We make a clone of the guide algorithm for each active interval. If the current interval is split into f active subintervals, $f-1$ new clones are created. Naturally, we create at most κ clones in our process. There is only one critical parameter value to determine the comparisons in a "usual" parametric search, and such a value can be found if we run the decision algorithm $O(\log N)$ times if the guide algorithm is a parallel algorithm on N processors. In our case, we run the decision algorithm $O(\kappa\log N)$ times at each level. Thus, we obtain the following theorem:

Theorem 3.5 *All the peaks on $\mathcal{L}_k$ within an interval I can be computed in $O(n\log n)+\tilde{O}((\kappa n)^{2/3})$ time if I has κ local peaks.*

Proof The parametric searching method gives $\tilde{O}(\kappa q(n,m))$ time complexity apart from the $O(m)$ preprocessing time. We balance $\kappa q(n,m)=\tilde{O}(\kappa n/m^{1/2})$ and m to have $m=(\kappa n)^{2/3}$. If $(\kappa n)^{2/3}<n$, we instead use $m=n\log n$. This gives the time complexity. □

Since $\kappa\leq 2k$, we have the following:

Corollary 3.6 *All the peaks on $\mathcal{L}_k$ can be computed in $O(n\log n)+\tilde{O}((kn)^{2/3})$ time.*

If $k=\Omega(n)$, this is slower than $O(k^{1/3}n\log^2 k)$ time algorithm for computing whole k-level, since our logarithmic factor is larger than 2. However, it is better if $k=O(n^{1-\epsilon})$ or $\kappa=O(n^{1-\epsilon})$.

3.1 Computing selected peaks

When κ is large, it may be too expensive to compute all the local peaks. Suppose that we want to compute τ largest maximal peaks in the input interval I for $\tau<\kappa$ more efficiently than computing all the peaks. This can be done by combining Roos and Widmayer's algorithm and the algorithm given above.

We first run a binary search process with respect to y-coordinate value similar to Roos-Widmayer's algorithm. At the intersection of the arrangement with a horizontal line $y=y_0$, we compute intervals on the line which are below the k-level (we call them semi-active intervals) in $O(n\log n)$ time. We next compute the sum $s(y_0)$ of numbers of local peaks in the semi-active intervals. We could apply our counting method of Lemma 3 for each semi-active

intervals to compute the sum of maximal peaks in the intervals by using $O(\tau q(n,m))$ time. More simply, we can compute it in $O(n \log n)$ time by counting the number of intersecting positive slope lines with the horizontal line during the sweep. In precise, we also need information of the arrangement at endpoints of the input interval I if one (or both) of them is below k-level (we omit details).

If $s(y_0)$ is not between τ and 2τ, we continue binary search on y_0: If $s(y_0) > 2\tau$, we increase y_0 while if $s(y_0) < \tau$, we decrease y_0. Thus, we can eventually find a value y_0 such that $\tau \leq s(y_0) \leq 2\tau$. We have spent $O(n \log^2 n)$ time so far. Now, we search all peaks in the union of active intervals by using the method given in the previous section. The following theorem is easy to see:

Theorem 3.7 *We can compute τ largest maximal peaks of $\mathcal{L}_k$ in an interval I in $O(n \log^2 n) + \tilde{O}((\tau n)^{2/3})$ time. We can also compute τ largest local peaks (including both maximal and minimal peaks) in the same time complexity.*

Note that if we only use Roos and Widmayer's algorithm in a naive fusion to find τ largest peaks, it would cost $O(\tau n \log^2 n)$ time. Analogously, we can compute the τ-smallest minimal peaks.

Theorem 3.8 *We can compute τ smallest minimal peaks of $\mathcal{L}_k$ in an interval I in $O(n \log^2 n) + \tilde{O}((\tau n)^{2/3})$ time. We can also compute τ smallest local peaks (including both maximal and minimal peaks) in the same time complexity.*

For this problem, we can modify Roos and Widmayer's method [18] to compute them in $O(n \log^2 n + k\tau \log^2 k)$ time. Although $n + (\tau n)^{2/3} \leq 2(n + k\tau)$ always holds, the time complexity is better than the one in Theorem 3.1 by a polylogarithmic factor if $n^{1/2} \log^{-c} n < k < n^{1/2} \log^c n$ some constant c. Moreover, the algorithm does not use range search data structure, and hence much simpler.

Proposition 3.9 *The τ smallest local peaks in an interval I can be found in $O(n \log^2 n + k\tau \log^2 k)$ time.*

Proof First we search for a horizontal line h such that it intersects k-level, and the number of peaks below it is not less than τ. Such a line h can be found in $O(n \log^2 n)$ time. Next, let v and w be the leftmost intersection and the rightmost intersection with the k-level on h, and let J be the interval between them. If the k-level at an endpoint of the input interval I is below h, we connect the endpoint to J with a segment to form a chain C (with at most three segments). Let $\mathcal{H}_0$ be the set of lines in the arrangement intersecting with the chain C. The cardinality of $\mathcal{H}_0$ is at most $2k$ because of Lemma 3. Finally, we find all peaks below h by using the k-branching binary search. Here, by using the windowing method of [18], we should only take care of lines in $\mathcal{H}_0$ together with lines below endpoints of the chain C. There are at most $4k$ such lines. Hence, this second step can be done in $O(k\tau \log^2 k)$ time. □

4 Bottleneck edge length in a parametric spanning tree

Next, we consider the parametric spanning tree problem. Consider a connected graph $G = (V, E)$ with k nodes and n edges. For each edge $e \in E$, we associate a weight function $w_e(x)$, which is linear on a parameter x. We assume that the arrangement generated by lines $y = w_e(x) : e \in E$ is simple, i.e., no three lines intersect at a point. We can remove this assumption by giving a symbolic perturbation in general. We denote $G(x)$ for G if it is considered as a weighted graph with parametric weights. For a given value x, we consider the minimum spanning tree $T(x)$ of $G(x)$.

It is known that the transition of the structure of the minimum spanning tree $T(x)$ is $O(k^{1/3}n)$, and all the transitions can be computed in $O(kn \log n)$ time. Moreover, the average edge weight in the minimum spanning tree is a concave function in x, and the value of x maximizing the average edge weight of $T(x)$ can be computed in $O(n \log n)$ time [10, 13]. As parametric matroid problems, the average edge weight is a counterpart of the average of y-values of k lines below (or on) the k-level.

A natural counterpart of the k-level itself in the minimum spanning tree is the weight of the longest (i.e. maximum weight) edge in the minimum spanning tree. The edge is also called the *spanning bottleneck edge* at x ($SBE(x)$ in short), and its weight is denoted by $w_{SBE}(x)$. It is easily observed that $W_{SBE}(x)$ is the maximum value of w such that the subgraph of $G(x)$ constructed from the set of edges whose weights are less than w is not connected.

It is natural and important in sensitivity analysis to trace the trajectory $y = w_{SBE}(x)$ of the weight of $SBE(x)$. Analogously to the k-level, there are at most k maximal peaks in the trajectory $y = w_{SBE}(x)$. We want to compute peaks in the trajectory.

4.1 One-shot query for the longest edge in MST

We first consider efficient query for $SBE(x_0)$ for any given value of x_0 of the parameter. This query is called one-shot query for the SBE. A naive method is first construct $T(x_0)$ in $O(n)$ time, and select its longest edge. Instead, we use the Matoušek's set partition. In the dual space, the dual points of n weight functions of the edges are partitioned into r subsets of size $O(n/r)$. Each subset is contained in a triangle, and $O(n^{1/2})$ triangles are cut by any query line.

Accordingly, we partition the set of n edges into r subsets each has $O(n/r)$ edges. For each subset, we compute a spanning forest (irrelevant to edge weights) and store the connected components except singletons. Thus, each component has a forest with $O(\min\{k, n/r\})$ edges. This computation can be done in $O(n)$ additional time.

If we are given a parameter value x_0, we sort $O(r)$ vertices of the triangles with respect to the inner product of them with the vector $(x_0, 1)$. We do binary search on this sorting list. We guess a vertex v, and consider a line $\ell : y = x_0X + c$ which goes through v. We recognize the triangles which is below ℓ; thus, the edges in the subsets associated with the triangles has weights which is less than c. We construct a spanning forest F of the union of forests in these subsets: since they have $O(rk)$ edges, this can be done in $O(rk)$ time. If the forest F is a spanning tree, we decide v is possibly too large (in the sorting list), and continue the binary search.

Otherwise, we consider the subsets associated with the triangles cut by ℓ. They contains

$O(n/r^{1/2})$ points in total. We sort them with respect to the weights, and greedily insert into F until we have a spanning tree. If we have a spanning tree, return the longest edge in the tree. Note that the spanning tree is not a minimum spanning tree in general; however, we correctly recognize the longest edge in a minimum spanning tree. If we do not have a spanning tree, we decide v is too small, and continue the binary search.

This process needs $O(rk+n/r^{1/2})$ time, and we do this process $O(\log r)$ times during the binary search. Thus, the time complexity is $O(n^{2/3}k^{1/3}\log(n/k))$, which is slightly better than $O(n)$ if $k = o(n)$. By applying a hierarchical subdivision, we can further improve it: We fist start $r = r_1$, and decompose the subset of size $O(r_1 n)$ into r_2 smaller subsets, where $r_2 = r_1^{1/2}$, and we further continue the refinement for $r_i = r_{i-1}^{1/2}$ until r_i becomes below a constant. The query time becomes
$k(r_1 + r_1^{1/2}r_2 + \ldots + (r_1 r_2 \ldots r_{i-1})^{1/2} r_i) + n/(r_1 r_2 \ldots r_i)^{1/2}$. Setting $r_1 = (n/k)^{1/2}$, this enables $\tilde{O}((nk)^{1/2})$ time computation for $SBE(x_0)$. Similarly to the case of halfspace range searching, we can combine hierarchical cutting of the arrangement to have a space-time trade-off (we omit details in this version). Indeed, we have the following proposition:

Proposition 4.1 *If we spend $\tilde{O}(m)$ preprocessing time for $n \le m \le n^2/k$, we can do the one-shot query for SBE in $\tilde{O}(n/(m/k)^{1/2})$ time.*

Moreover, we will later use the following *two-shot reporting query* for a spanning forest, which reports a spanning forest consisting of edges whose weights function is below both of give two query points (x_0, y_0) and (x_1, y_1). This can be done similarly to one-shot query (this is a counterpart of the simplex range searching if the one-shot query is a counterpart of the halfplane range searching).

Proposition 4.2 *If we spend $\tilde{O}(m)$ preprocessing time for $n \le m \le n^2/k$, we can do the two-shot reporting query a spanning forest in $\tilde{O}(\{n/(m/k)^{1/2} + k\})$ time.*

4.2 Computing the maximum peak

Let us consider the problem of computing the maximum peak in I. First, we straightforwardly apply Roos-Widmayer's algorithm. For a given y-value y_0, we want to decide whether $Max_{t\in I} w_{SBE}(t) \le y_0$ or not. We dynamically update the spanning forest associated with edges with weight below y_0 from $t = x_0$ to $t = x_1$ if $I = [x_0, x_1]$. If we find a value $x \in I$ such that the spanning forest becomes a spanning tree, we know $Max_{t\in I} w_{SBE}(t) \le y_0$. It costs $O(k^{1/2})$ time to update a minimum spanning forest for insertion and deletion of edges. The line $y = y_0$ has at most n intersections with lines associated with weight functions, and hence the method needs $O(nk^{1/2})$ time for the decision. Thus, the maximum peak can be found in $O(nk^{1/2}\log^2 n)$ time.

We try to improve the above time complexity. We subdivide the line $y = y_0$ into $\lceil n/s \rceil$ intervals such that each interval contains at most s intersection points. For each interval $I_i = [x_i, x_{i+1}]$, we perform the two-shot reporting query at (x_0, y_0) and (x_1, y_0). The reported forest F is constructed from edges whose weight is less than y_0 both at $x = x_i$ and $x = x_{i+1}$. If the forest has more than $s+1$ connected components, it is impossible that $w_{SBE}(t) \le y_0$ for a $t \in I_i$. Otherwise, we dynamically maintain the spanning tree, where we contract nodes into at most $s+1$ super nodes each of which associate with a connected component of the

forest F. Our graph has only s edges, and hence the update can be done $O(\sqrt{s})$ time per intersection.

Hence, total time complexity becomes $\tilde{O}(n\sqrt{s} + (n/s)[\{n/(m/k)^{1/2}\} + k] + m)$. If we optimize this, we have $\tilde{O}(n^{8/7}k^{1/7} + nk^{1/3})$. This is an improvement over $O(nk^{1/2})$, since $n = O(k^2)$.

4.3 Computing all local peaks for SBE

It is desired to apply the method of computing all peaks in k levels to SBE in a parametric graph. It is known that there are at most k maximal peaks in $y = w_{SBE}(x)$. Unfortunately, we have no good method to know the number of peaks within an interval $I = [x_0, x_1]$ exactly, since we do not have a property that is a counterpart of Lemma 3. The only known method for the authors is to compute the minimum spanning trees $T(x_0)$ and $T(x_1)$ explicitly, and compute the difference $d(I)$ between the number of edges whose weight functions are positive slopes. For any disjoint set of intervals, the sum of $d(I)$ over the intervals is at most k, and $d(I)$ gives an upper bound of maximal peaks of $y = w_{SBE}(x)$within I. However, it is often an overestimate, since $d(I)$ gives the number of peaks of k trajectories of weights of all edges (not only maximum one) in the parametric minimum spanning tree.

By using the above observation, we have an $\tilde{O}(kf(n,k))$ time algorithm if we have an algorithm to compute $T(x_0)$ for a given x_0 in $O(f(n,k))$ time. Unfortunately, we only have $f(n,k) = O(n)$, which leads to an $\tilde{O}(kn)$ time algorithm, which is inferior to a known algorithm to compute all transitions of the parametric minimum spanning tree. We can compute τ largest peaks in transitions of edge weights in MST in $\tilde{O}(\tau n)$ time, if we include all peaks of all edges in MST; however, the number of peaks appeared at the transitions of the longest edge among them may be much smaller than τ.

Although the above method is not attractive for the minimum spanning tree, the method is applicable to any parametric matroid, and hence it is useful if we do not have a dynamic algorithm to maintain a minimum weight base, since the current $O(kn \log n)$ time algorithm to compute the transitions of parametric minimum spanning tree needs $O(k^{2/3})$ time method (indeed, it can be done in $O(k^{1/2})$ time) to update a minimum spanning tree.

5 Concluding remarks

The number of maximal peaks in a k-level is known to be at most ${}_kC_d$ if we have d dimensions [5]. Hence, this is much smaller than the complexity of whole arrangement, especially if k is much smaller than n. However, to the author's knowledge, the problem of computing peaks in the k-level for a higher dimensional arrangement is open. Although the algorithm of Roos and Widmayer can be applied to 3-dimensional case, it needs $\tilde{O}(n^2)$ time to compute the largest peak (i.e. global maximum) if we naively implement it. One necessary constituent is to develop a counterpart of Lemma 3: Given an arrangement of n hyperplanes in the three-dimensional space, preprocess it, and for any given three points A, B, and C in the plane $z = 0$, decide whether the triangle ABC contains (a projection) of a peak in the k-level or not efficiently. For the purpose, we probably need a counterpart of Lemma 3: Give a criterion of the existence of a peak from the information of the set of hyperplanes below k level at each of A, B, and C. In two-dimensional space, the lines are classified into positive

slope lines and negative slope lines, while this kind of natural discrete classification of planes in the space does not exist. Moreover, it is difficult to find a counterpart of concave chain decomposition for three dimensional k-level [14]. These lacks make the problem difficult, although authors think it is quite attractive.

References

[1] P. Agarwal, B. Aronov, T. Chan, and M. Sharir, "On Levels in Arrangement of Lines, Segments, Planes, and Triangles," *Discrete & Comput. Geom.* 19 (1998), pp. 315-331.

[2] J. Bash, L. Guibas, and H. Hershberger, "Data Structures for Mobile Data," *Proc. 8th ACM-SIAM Symp. on Disc. Alg.* (1997), pp. 747-756.

[3] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, "An Optimal Algorithm for Slope Selection," *SIAM J. Comput.* **18-4** (1989), pp. 792-810.

[4] B. Chazelle, "Cutting Hyperplanes for Divide-And-Conquer", *Discrete & Computational Geometry* **9** (1993) pp. 145–158.

[5] K. Clarkson, "A Bound on Local Minima of Arrangement That Implies the Upper Bound Theorem," *Discrete & Comput. Geom.* 10 (1993), pp. 427-433.

[6] T. Dey, "Improved Bound on Planar K-Sets and Related Problems," *Discrete & Comput. Geom.* 19 (1998), pp. 373-383.

[7] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, ETACS Monographs on TCS 10, Springer-Verlag, 1987.

[8] D. Eppstein, "Geometric Lower Bounds for Parametric Matroid Optimization," *Discrete & Comput. Geom.* (1998), pp. 463-476.

[9] D. Eppstein, Z. Galil, G.F.Italiano, and A. Nissenzweig, "Sparcification–A Technique for Speeding Up Dynamic Graph Algorithms" *J. ACM* **44-5** (1997), pp. 669-696.

[10] D. Fernandez-Baca, G. Slutki, and D. Eppstein, " Using Sparcification for Parametric Minimum Spanning Tree Problems", *Nordic J. Computing* **3-4** (1996) pp. 352-366.

[11] D. Gusfield, "Bound for the Parametric Spanning Tree Problem," *Proc. Humbolt Conf. Graph Theory, Combinatorics, and Computing* (1979), pp. 173-183.

[12] D. Gusfield, *Sensitivity Analysis for Combinatorial Optimization,* Ph. D. Thesis, Memorandum No. UCB/ERL M80/22, U.C. Berkeley, 1980.

[13] N. Katoh, K. Iwano, and T. Tokuyama, "On Minimum and Maximum Spanning Trees of Linearly Moving Points," *Discrete & Comput. Geom.* 13 (1995) pp. 161-176.

[14] N. Katoh, H. Tamaki and T. Tokuyama, "Parametric Polymatroid Optimization and Its Geometric Applications," *Proc. 10th ACM-SIAM SODA* (1999) pp. 517-526.

[15] N. Katoh and T. Tokuyama, "Lovász's Theorem for the K-level of an Arrangement of Concave Surfaces and Its Applications," to appear in *Proceedings of 40th IEEE FOCS* (1999).

[16] J. Matoušek, "Efficient Partition Trees," *Discrete & Comput. Geom.* (1992) 315–334.

[17] N. Megiddo, "Applying Parallel Computation Algorithms," in the design of serial algorithms, *J. ACM* **30** (1983) 852–865.

[18] T. Roos and P. Widmayer, "K-violation Linear Programming," *Information Processing Letters* **52** (1994) 109-114.

[19] J. Salowe, Parametric Search, Section 37 of *Handbook of Discrete and Computational Geometry* (1997) 683–695, CRC Press.

[20] G. Tóth, "Point Sets with Many k-sets," *Proc. 16th SOCG* (2000), pp. 37-42.

[21] Web page on k-set, dissecting lines, and parametric optimization: http://liinwww.ira.uka.de/searchbib/Theory/kset.